EMBEDDED OPEN SOURCE SUMMIT

EMBEDDED LINUX CONFERENCE

# Display Sharing in heterogeneous systems using Linux as high level OS

**Date: 2024-04-17**

**Location: Seattle**

**Devarsh Thakkar**

**Soumya Tripathy**

# About us: TI Processors and Open source





Decades of contribution and collaboration

Ingrained culture to give back to the community


**Upstream FIRST!**


Focus on long term, sustainable and quality products



Upstream and opensource ecosystem in device architecture




Upstream FIRST mentality!

# Authors

**Devarsh Thakkar, Embedded Software Engineer** *at Texas Instruments, Bangalore.*

Devarsh Thakkar works as an Embedded Linux developer at Texas Instruments. He has 10+ years of experience in software development ranging from open-source bootloaders to the Linux kernel, middleware frameworks and applications. His expertise lies in Audio/Video related multimedia frameworks, Linux media subsystems, Linux device drivers and applications. He has made contributions to open-source projects such as U-boot , Linux Kernel and Gstreamer and also presented in international conferences.

**Soumya Tripathy, Embedded Software Engineer** *at Texas Instruments, Bangalore.*

Soumya has been working with TI for 2 years with contributions and expertise in the field of bootloader, flash devices and display controller for the Sitara family of processors. Prior to TI, he worked as a firmware developer at Robert Bosch. He made contributions to firmware development of Servo drives by Bosch Rexroth related to the field of Industrial communication and factory automation products.

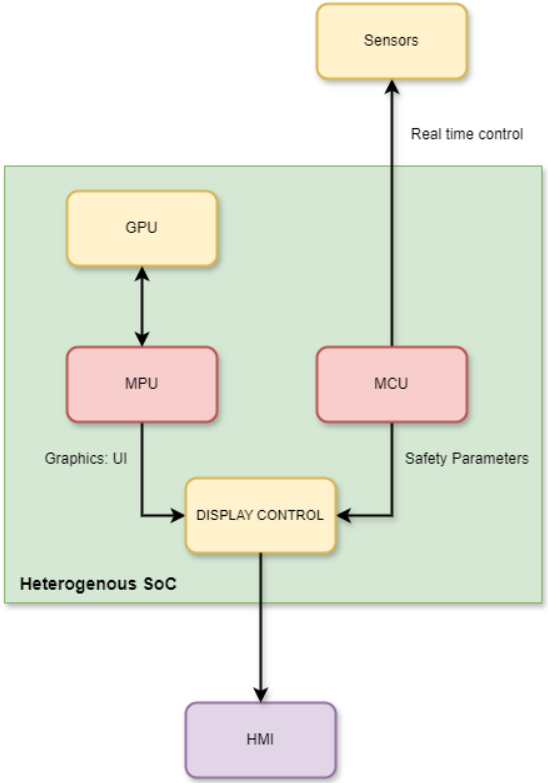
# Overview

- Introduction
  - Practical Applications
  - Problem Statement
- Possible Approaches
  - IPC based
    - RPMSG_KDRV
    - RPI_KMS
  - Non-IPC based – Static partitioning
    - TI-DSS
    - Examples with static partitioning
    - Firewalling
- Demo : Display Sharing
- References
- Credits & Acknowledgement
- Q&A

# Introduction

# Introduction

- What is display sharing ?
  - Display controller functionality being used by more than one processing entities.
- Where ?
  - Heterogenous SoCs with MPU and MCU together.
  - Display can be shared between MCU and MPU.
- Why ?
  - RTOS controlled displays are fast and low-latency.
  - Mostly lack the capability of GPU based rendering : application core renders graphics.
  - RTOS controls safety parameters (tell-tales in a display cluster or safety industrial params)
  - Safety params should remain unhindered in case of application core crash.



# Practical Applications


Automotive Display Cluster





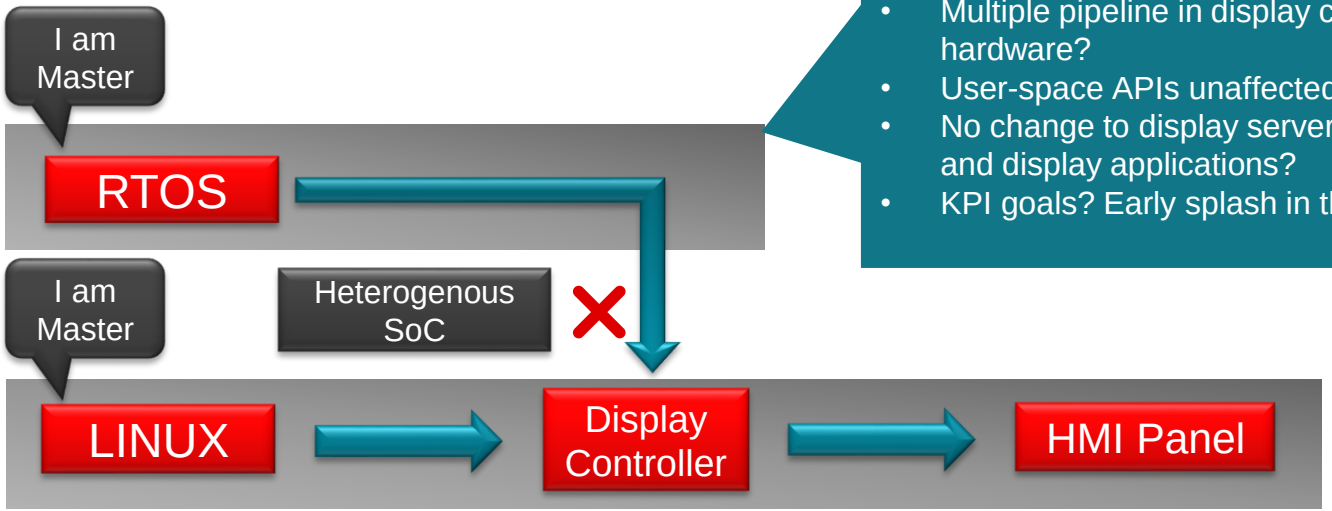

Industrial HMI




Medical Instruments

# Problem Statement

- Heterogenous SoC : multiple entities access same display controller.
- Linux assumes full control.
- No upstream/standard framework to share display between processing cores.
- It's a generic problem !!!
- Vendors having custom solutions to same problem.




- How to define who controls the global display configuration ?
- How to share display initialization and display configuration info with other hosts ?
- How to partition display resources between different hosts ?
- How to protect display context of one host from corruption due to another host ?
- Multiple pipeline in display controller? Leverage hardware?
- User-space APIs unaffected?
- No change to display server based windowing systems and display applications?
- KPI goals? Early splash in the system

# Possible Approaches

# Possible Approaches

Not in Upstream

RFC posted

## IPC BASED

- RPMSG_KDRV
- RPI_KMS

## NON - IPC BASED

- STATIC PARTIONING

# Possible Approaches

# IPC BASED

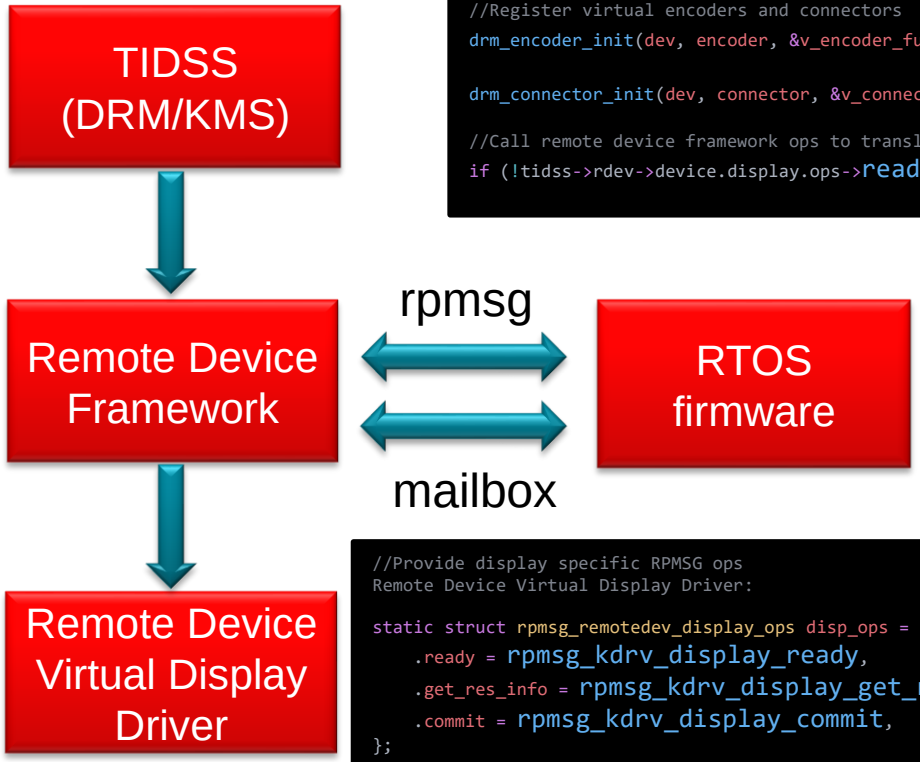
# IPC based

- Display controller's power domain, clock domains and register space being controlled by only one processing core which acts as display server
- The other processing entities which is acting as a client to display controller can request specific display resources (for e.g. video plane) from the processing core using IPC (mailbox, rpmsg).
- Handshake between display server and display client are on majorly below aspects :

| | |
|---|---|
| Display Ready | • Display client know that display is ready and initialized by the core acting as display server. |
| Current Mode | • Display client know what is the current mode set by the display server. |
| EDID Info | • Help the client know the list of all supported video modes. (optional) |
| Frame Buffer Commit | • Display client submit framebuffer for display to display server. |
| Vblank Interrupts | • Dedicated IRQs, interrupt forwarding from firmware, hrtimer based polling. |

# IPC based – RPMSG_KDRV

- RPMSG_KDRV : rpmsg based device virtualization framework using rpmsg_kdrv bus
- API's to talk to send request and receive response from remote.

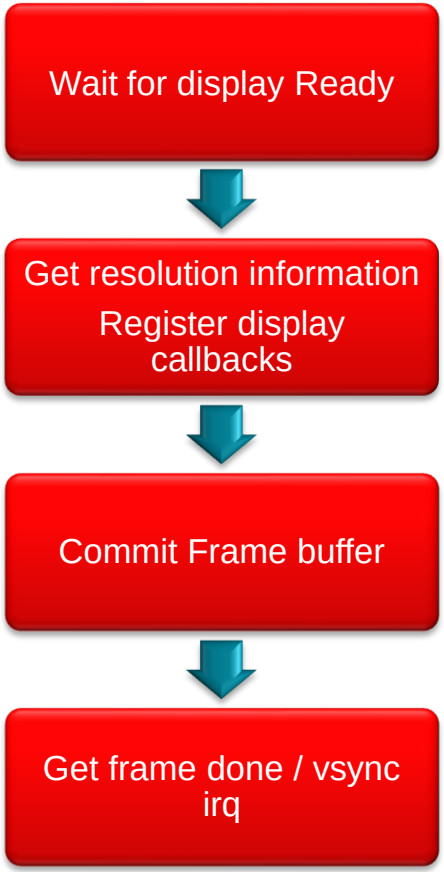



```
//Register virtual encoders and connectors
drm_encoder_init(dev, encoder, &v_encoder_funcs, DRM_MODE_ENCODER_VIRTUAL, "encoder-0x%x", id);

drm_connector_init(dev, connector, &v_connector_funcs, DRM_MODE_CONNECTOR_VIRTUAL);

//Call remote device framework ops to translate DRM/KMS settings to RPMSG and send to remote dev
if (!tidss->rdev->device.display.ops->ready(tidss->rdev)) {
```

```
//Provide display specific RPMSG ops
Remote Device Virtual Display Driver:

static struct rpmsg_remotedev_display_ops disp_ops = {
    .ready = rpmsg_kdrv_display_ready,
    .get_res_info = rpmsg_kdrv_display_get_res,
    .commit = rpmsg_kdrv_display_commit,
};
```

```
//Send RPMSG with callback function info which needs to be called when
response is received
int rpmsg_kdrv_send_request_with_callback(struct
rpmsg_device *rpdev,
                uint32_t device_id, void *message, uint32_t message_size,
void *cb_data, request_cb_t callback);

//Send RPMSG with response
int rpmsg_kdrv_send_request_with_response(struct
rpmsg_device *rpdev,
                uint32_t device_id, void *message, uint32_t message_size,
                void *response, uint32_t response_size);

//Send RPMSG
int rpmsg_kdrv_send_message(struct rpmsg_device *rpdev,
                uint32_t device_id, void *message, uint32_t message_size);
```

# IPC based – RPMSG_KDRV



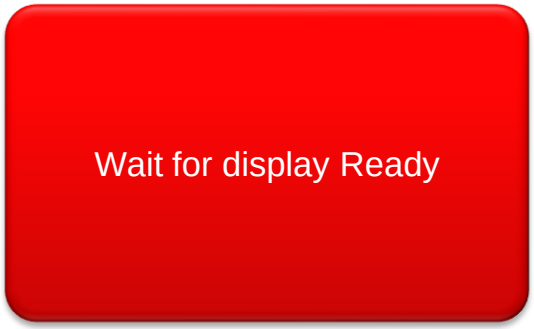

```
tidss_probe {
      /* Cant really do much if the remotedev is not ready yet */
          if (!tidss->rdev->device.display.ops->ready(tidss->rdev)) {
              ret = -EPROBE_DEFER;
              goto disconnect;
}
```

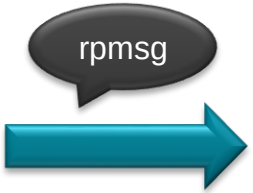
```
struct rpmsg_remotedev_display_cb tidss_rdev_cb  = {
       .commit_done = v_crtc_commit_done,
       .buffer_done = v_crtc_buffer_done,
};
int tidss_modeset_init(struct tidss_device *tidss)
{
    if (tidss->rdev) {
            tidss->rdev->device.display.ops->get_res_info(tidss->rdev, &tidss->rres);

            if (tidss->rres.num_disps)
                    tidss->rdev->device.display.cb_ops = &tidss_rdev_cb;
}
```

```
static void v_crtc_flush_to_remote() {

 v_crtc->tidss->rdev->device.display.ops->commit(v_crtc->tidss->rdev, commit);

}
```

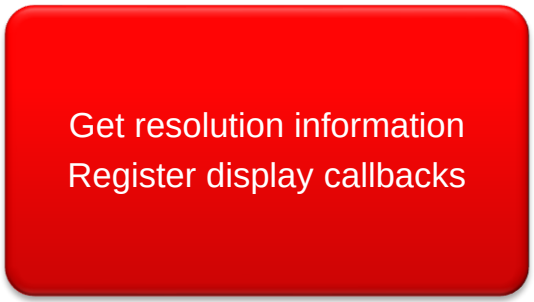
```
void v_crtc_commit_done(struct rpmsg_remotedev_display_commit *commit, void *data)
{
        struct drm_crtc *crtc = commit->priv;
        v_crtc_vblank_irq(crtc);
        kfree(commit);
}
```

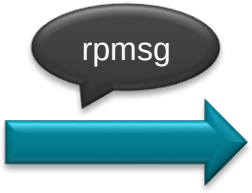
# IPC based – RPMSG_KDRV




```
//Wait for display Ready

static bool rpmsg_kdrv_display_ready() {

    req->header.message_type = RPMSG_KDRV_TP_DISPLAY_READY_QUERY_REQUEST;

    rpmsg_kdrv_send_request_with_response

    retval = resp->ready ? true : false;
}
```

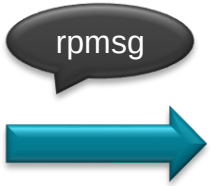



```
//Get Display resolution and register callback
static int rpmsg_kdrv_display_get_res() {

    req->header.message_type = RPMSG_KDRV_TP_DISPLAY_RES_INFO_REQUEST;

}

rpmsg_kdrv_send_request_with_response(rpdev, kddev->device_id, req,
sizeof(*req), resp, sizeof(*resp));
```

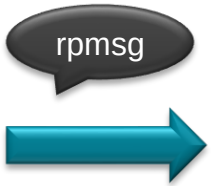
# IPC based – RPMSG_KDRV

Commit Frame buffer



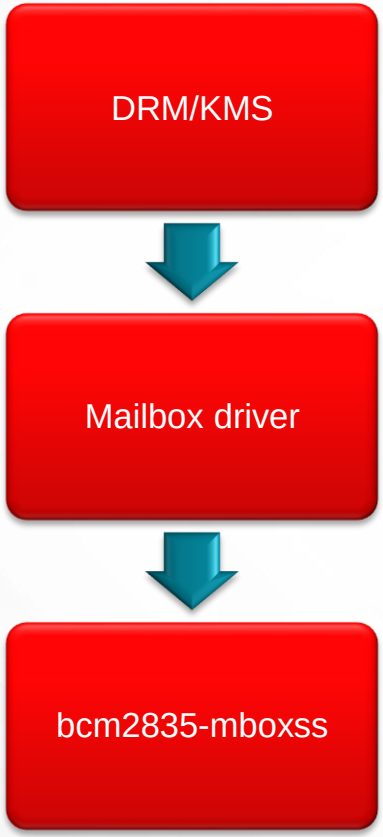

```
//Commit frame buffer
rpmsg_kdrv_display_commit ()
{
    req->header.message_type = RPMSG_KDRV_TP_DISPLAY_COMMIT_REQUEST;

        if (!rpmsg_kdrv_display_copy_commit(kddev, req, commit)) {
                dev_err(&kddev->dev, "%s: failed to copy commit request\n", __func__);
                ret = -ENOMEM;
                goto out;
        }
        ret = rpmsg_kdrv_send_request_with_response(rpdev, kddev->device_id, req, sizeof(*req),
                        resp, sizeof(*resp));
        if (ret) {
                dev_err(&kddev->dev, "%s: rpmsg_kdrv_send_request_with_response\n", __func__);
                goto nosend;
        }
}
```

Get frame done / vsync irq




```
//Get frame done irq
static void rpmsg_kdrv_display_handle_commit()
{
        if (rdev->device.display.cb_ops && rdev->device.display.cb_ops->commit_done)
                rdev->device.display.cb_ops->commit_done(commit, rdev->cb_data);
}

static int rpmsg_kdrv_display_callback(struct rpmsg_kdrv_device *dev, void *msg, int len)
{
        struct rpmsg_kdrv_display_message_header *hdr = msg;
        if (hdr->message_type == RPMSG_KDRV_TP_DISPLAY_COMMIT_DONE_MESSAGE)
                rpmsg_kdrv_display_handle_commit(dev, msg);
        else if (hdr->message_type == RPMSG_KDRV_TP_DISPLAY_BUFFER_DONE_MESSAGE)
                rpmsg_kdrv_display_handle_buffer(dev, msg);
        return 0;
}
```

# IPC based – Rpi FKMS




```
enum rpi_firmware_property_tag {

/* Dispmanx TAGS */
    RPI_FIRMWARE_FRAMEBUFFER_ALLOCATE =                   0x00040001,
    RPI_FIRMWARE_FRAMEBUFFER_BLANK =                      0x00040002,
    RPI_FIRMWARE_FRAMEBUFFER_GET_PHYSICAL_WIDTH_HEIGHT =  0x00040003,
    RPI_FIRMWARE_FRAMEBUFFER_GET_VIRTUAL_WIDTH_HEIGHT =   0x00040004,
    RPI_FIRMWARE_FRAMEBUFFER_GET_DEPTH =                  0x00040005,
    RPI_FIRMWARE_FRAMEBUFFER_GET_PIXEL_ORDER =            0x00040006,
    RPI_FIRMWARE_FRAMEBUFFER_GET_ALPHA_MODE =             0x00040007,
    RPI_FIRMWARE_FRAMEBUFFER_GET_PITCH =                  0x00040008,
    RPI_FIRMWARE_FRAMEBUFFER_GET_VIRTUAL_OFFSET =         0x00040009,
    RPI_FIRMWARE_FRAMEBUFFER_GET_OVERSCAN =               0x0004000a,
    RPI_FIRMWARE_FRAMEBUFFER_GET_PALETTE =                0x0004000b,
    RPI_FIRMWARE_FRAMEBUFFER_GET_LAYER =                  0x0004000c,
    RPI_FIRMWARE_FRAMEBUFFER_GET_TRANSFORM =              0x0004000d,
    RPI_FIRMWARE_FRAMEBUFFER_GET_VSYNC =                  0x0004000e,
};

ret = devm_request_irq(dev, platform_get_irq(pdev, 0),
                       vc4_crtc2712_irq_handler, 0,
                       "vc4 firmware kms", crtc_list);


rpi_firmware_transaction()
{
    ret = mbox_send_message(fw->chan, &message);
}
```

Message tags for firmware

Request Irq

Message to firmware over mailbox

# IPC based – Rpi FKMS

Get display config

```
ret = rpi_firmware_property(vc4->firmware,
                    RPI_FIRMWARE_GET_DISPLAY_CFG,
                    &fkms->cfg, sizeof(fkms->cfg));

for (display_num = 0; display_num < num_displays; display_num++) {
    display_id = display_num;
    ret = rpi_firmware_property(vc4->firmware,
                        RPI_FIRMWARE_FRAMEBUFFER_GET_DISPLAY_ID,
                        &display_id, sizeof(display_id));
```

Initialize plane info

```
  vc4_fkms_plane_init{

        vc4_plane->mb.tag.tag = RPI_FIRMWARE_SET_PLANE;
    vc4_plane->mb.tag.buf_size = sizeof(struct set_plane);
    vc4_plane->mb.tag.req_resp_size = 0;
    vc4_plane->mb.plane.display = display_num;
    vc4_plane->mb.plane.plane_id = plane_id;
    vc4_plane->mb.plane.layer = default_zpos ? default_zpos : -127;
}

struct mailbox_set_plane blank_mb = {
        .tag = { RPI_FIRMWARE_SET_PLANE, sizeof(struct set_plane), 0 },
        .plane = {
            .display = vc4_plane->mb.plane.display,
            .plane_id = vc4_plane->mb.plane.plane_id,
        }
    };

if (blank)
        ret = rpi_firmware_property_list(vc4->firmware, &blank_mb,
                        sizeof(blank_mb));
```

Power on display

```
Power  on display
static void vc4_fkms_display_power(struct drm_encoder *encoder, bool power)
{
    struct vc4_fkms_encoder *vc4_encoder = to_vc4_fkms_encoder(encoder);
    struct vc4_dev *vc4 = to_vc4_dev(encoder->dev);

    struct mailbox_display_pwr pwr = {
        .tag1 = {RPI_FIRMWARE_SET_DISPLAY_POWER, 8, 0, },
        .display = vc4_encoder->display_num,
        .state = power ? 1 : 0,
    };

    rpi_firmware_property_list(vc4->firmware, &pwr, sizeof(pwr));
}
```

Set display mode and timing

```
static void vc4_crtc_mode_set_nofb(struct drm_crtc *crtc)
{
    struct mailbox_set_mode mb = {
        .tag1 = { RPI_FIRMWARE_SET_TIMING,
              sizeof(struct set_timings), 0},
    };
    mb.timings.clock = mode->clock;
    mb.timings.hdisplay = mode->hdisplay;
    mb.timings.hsync_start = mode->hsync_start;
    mb.timings.hsync_end = mode->hsync_end;
    mb.timings.htotal = mode->htotal;
    mb.timings.hskew = mode->hskew;
    mb.timings.vdisplay = mode->vdisplay;
    mb.timings.vsync_start = mode->vsync_start;
    mb.timings.vsync_end = mode->vsync_end;
    mb.timings.vtotal = mode->vtotal;
    mb.timings.vscan = mode->vscan;
    mb.timings.vrefresh = drm_mode_vrefresh(mode);
    mb.timings.flags = 0;

    ret = rpi_firmware_property_list(vc4->firmware, &mb, sizeof(mb));
}
```

# IPC: Further Thoughts (from upstream POV)

## rpmsg_kms driver

- Sits on top of rpmsg_bus
- Leverage existing rpmsg framework (virtio_rpmsg, rpmsg_ns)
- Similar to rpmsg_tty

## virtio_kms driver

- Sits on top of virtio bus
- Can share common ground with display virtualization (virtio_gpu)
- Similar to virtio_console : remote device support can be added

# Possible Approaches

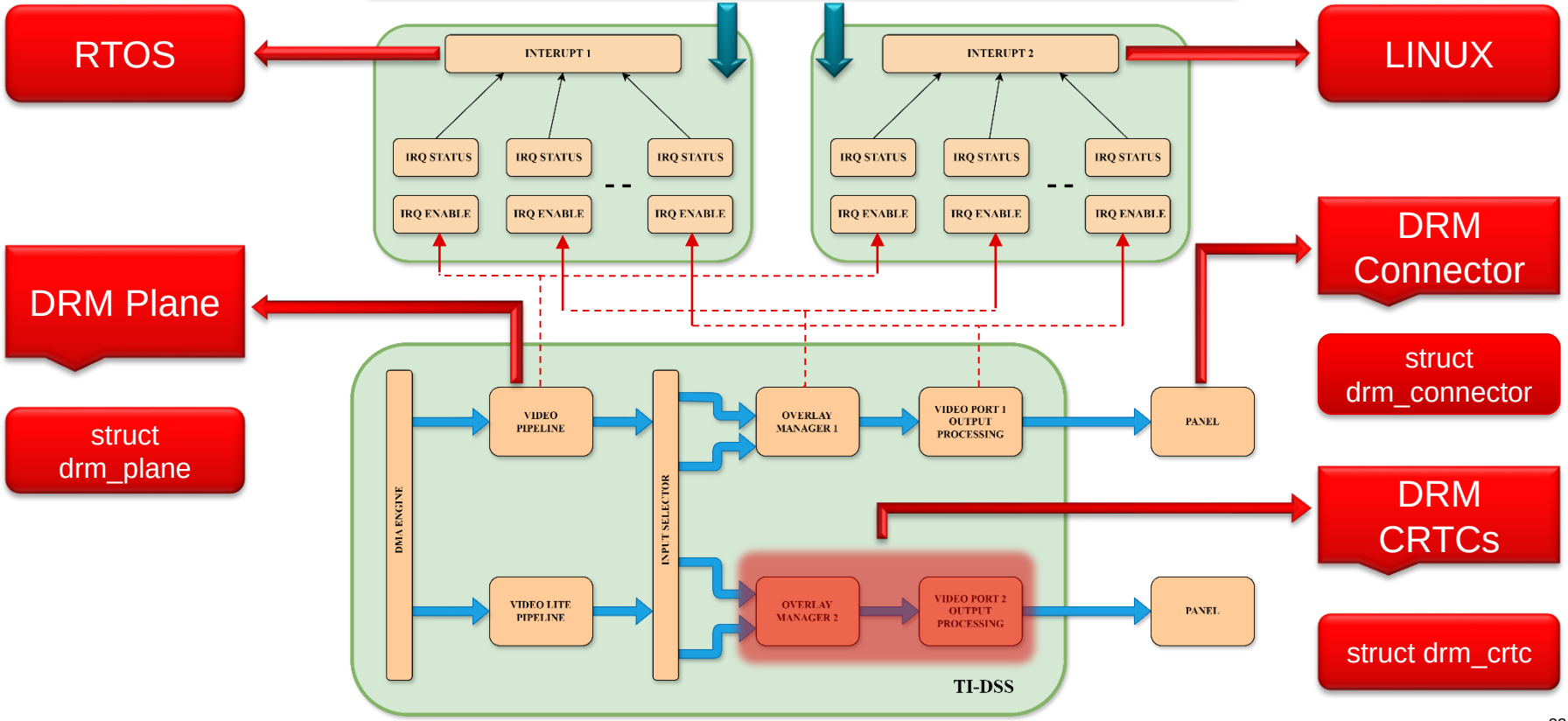
# NON - IPC BASED STATIC PARTIONING

# Static Partitioning of display resources

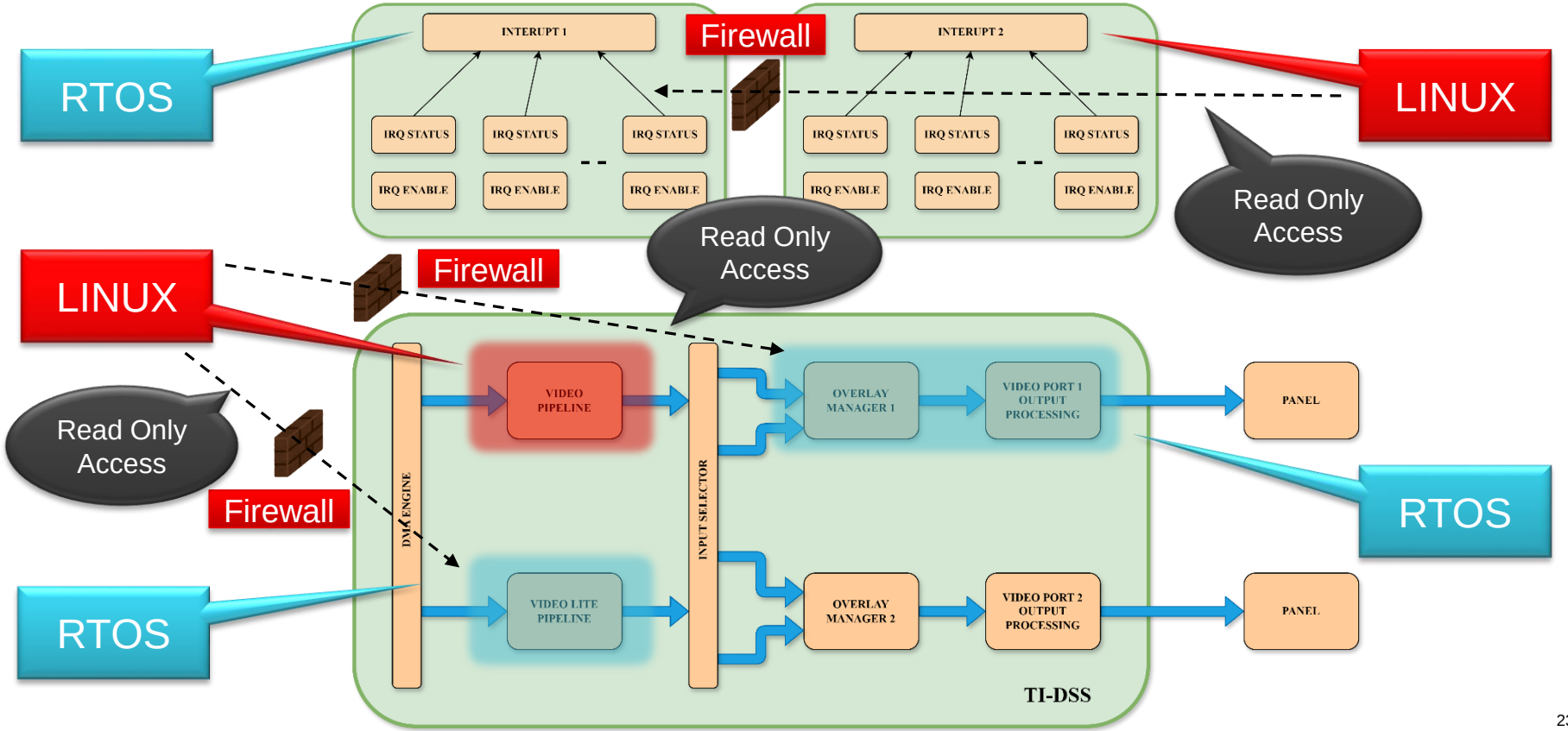
- Partition the display resources between multiple hosts
  - Each host binary is compiled with support for only the display resources controlled by it.
- Display hardware needs to have separate register space and separate interrupt for each host
  - Each host to use the display resources owned by it independently.
- One core acts as display master
  - Initializes the display hardware and manages display global register space.

# TI-DSS

Separate register space : Common register region
Perform global configuration, subscribe to Vblank interrupts



# Static Partitioning : TI-DSS



# Static Partitioning in Device Tree

- Define attributes for device tree to share or own display resources between multiple entities.
- Each of the display resource is given attributes related to **sharing** and **ownership.**

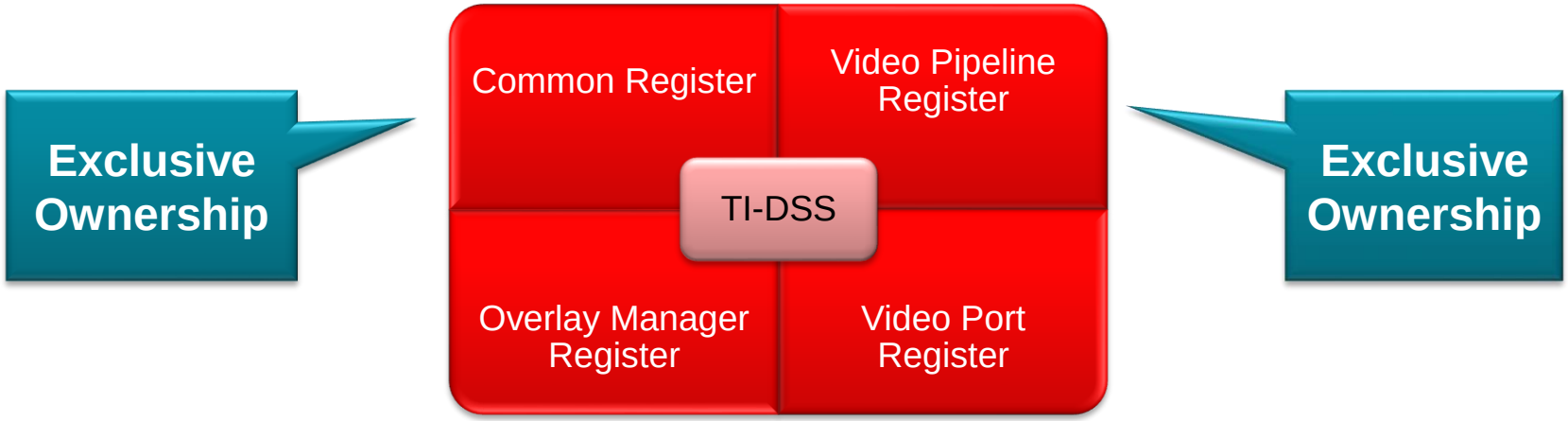
**Exclusive Ownership Attribute**

- Exclusivity for a display resource.

**Shared Mode Attribute**

- Shares the resource with a different host or entity.

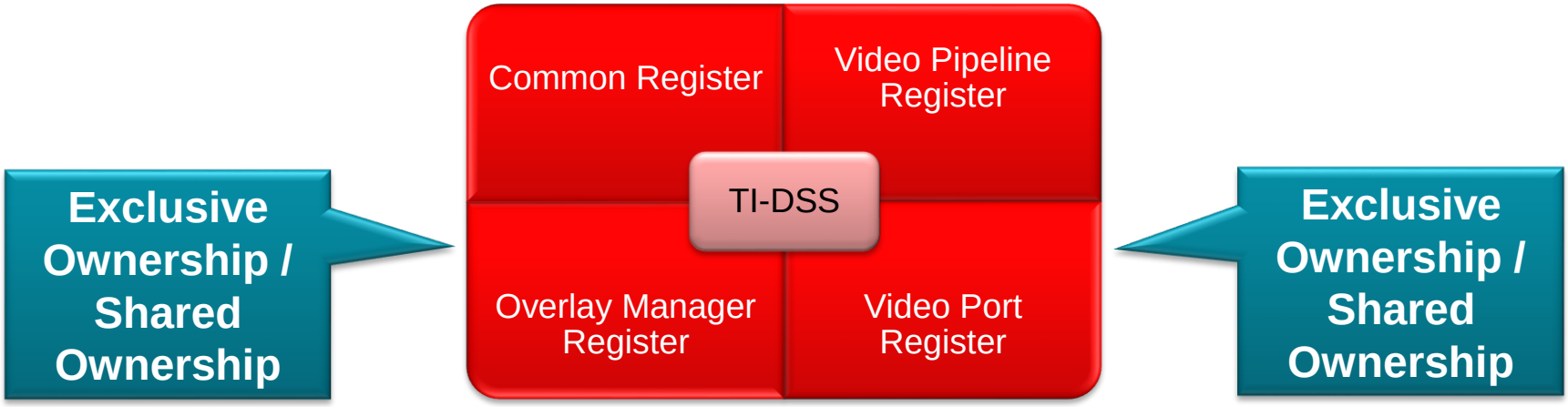
# Static Partitioning in Device Tree




- Each host has separate register region for controlling the display.
- Only one host has global configuration support.
- Processing core can own corresponding register region. (Other regions are invisible to it.)

- Exclusively owned by one core.
- Remains invisible to other processing cores.

# Static Partitioning in Device Tree




- Multiple video pipelines are connected to it.
- Resource can be exclusively owned with write access.
- Resource can be used in shared mode, when owned by other processing core. (Processing cores utilizing any of the video pipelines controlled by this overlay manager).

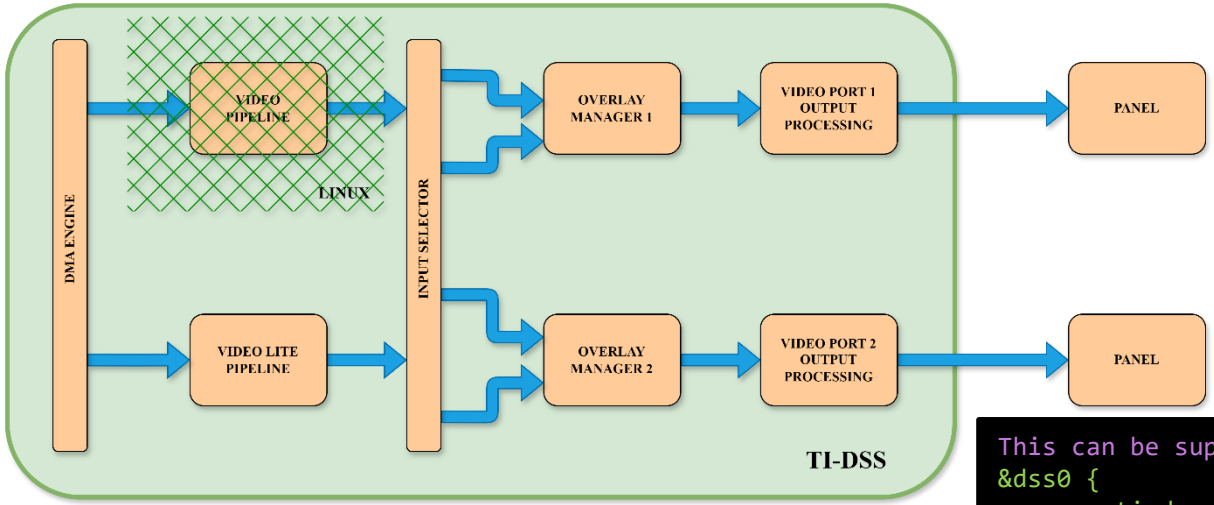
- Similar to video overlay manager.
- Resource can be exclusively owned with write access or in shared mode.

# Static Partitioning in Device Tree

- Below device-tree properties are made available to user so that they can customize and tailor their sharing solution and Linux configures the display pipeline accordingly.

| | |
|---|---|
| **ti,dss-shared-mode** | • Enable display sharing mode. |
| **ti,dss-shared-mode-planes** | • Display planes owned by Linux. |
| **ti,dss-shared-mode-vp** | • Video Port being used to control above planes. |
| **ti,dss-shared-mode-owned-vp** | • Specify which of the Video Ports are owned by Linux , only the core having ownership has write access to Video Port. |
| **ti,dss-shared-mode-common** | • Specify which of the common region should be used by Linux for programming the DSS. |

# Example 1: Linux owning one video pipeline with RTOS controlling the rest of DSS.



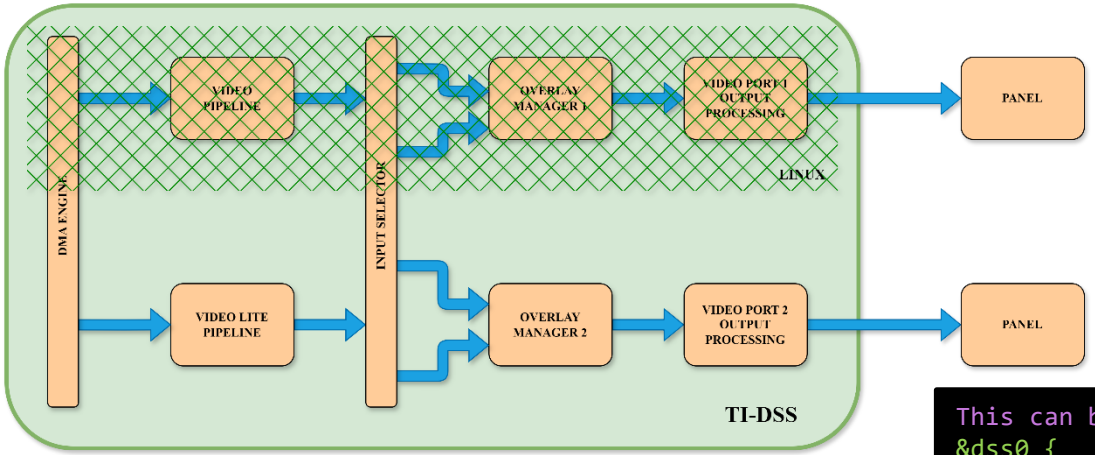

```
This can be supported with below configuration :
&dss0 {
        ti,dss-shared-mode;
        ti,dss-shared-mode-vp = "vp1";
        ti,dss-shared-mode-vp-owned = <0>;
        ti,dss-shared-mode-common = "common1";
        ti,dss-shared-mode-planes = "vid";
        ti,dss-shared-mode-plane-zorder = <0>;
        interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>;
};
```

# Example 1: Linux owning one video pipeline with RTOS controlling the rest of DSS.


```
const struct dispc_features dispc_am62p51_feats = {
    .common = "common",
    .common_regs = tidss_am65x_common_regs,

    .num_vps = 2,
    .vp_name = { "vp1", "vp2" },
    .ovr_name = { "ovr1", "ovr2" },
    .vpclk_name =  { "vp1", "vp2" },


    .num_planes = 2,
    /* note: vid is plane_id 0 and vidl1 is plane_id 1 */
    .vid_name = { "vid", "vidl1" },
    .vid_lite = { false, true, },
    .vid_order = { 1, 0 },

    /* 3rd output port is not representative of a 3rd pipeline */
    .num_outputs = 3,
    .output_type = { DISPC_OUTPUT_OLDI, DISPC_OUTPUT_DPI, DISPC_OUTPUT_OLDI, },
    .output_source_vp = { 0, 1, 0, },
};
```

```
const struct dispc_features dispc_am62p51_feats = {

    .common = "common",
    .common_regs = tidss_am62_common1_regs,

    .num_vps = 0,
    .vp_name = { "vp1",},
    .ovr_name = { "ovr1",},
    .vpclk_name =  { "vp1",},

    .num_planes = 1,
    /* note: vid is plane_id 0 and vidl1 is plane_id 1 */
    .vid_name = { "vid",},
    .vid_lite = { false },
    .vid_order = { 0 },

    /* 3rd output port is not representative of a 3rd pipeline */
    .num_outputs = 1,
    .output_type = { DISPC_OUTPUT_OLDI,},
    .output_source_vp = { 0,},
};
```

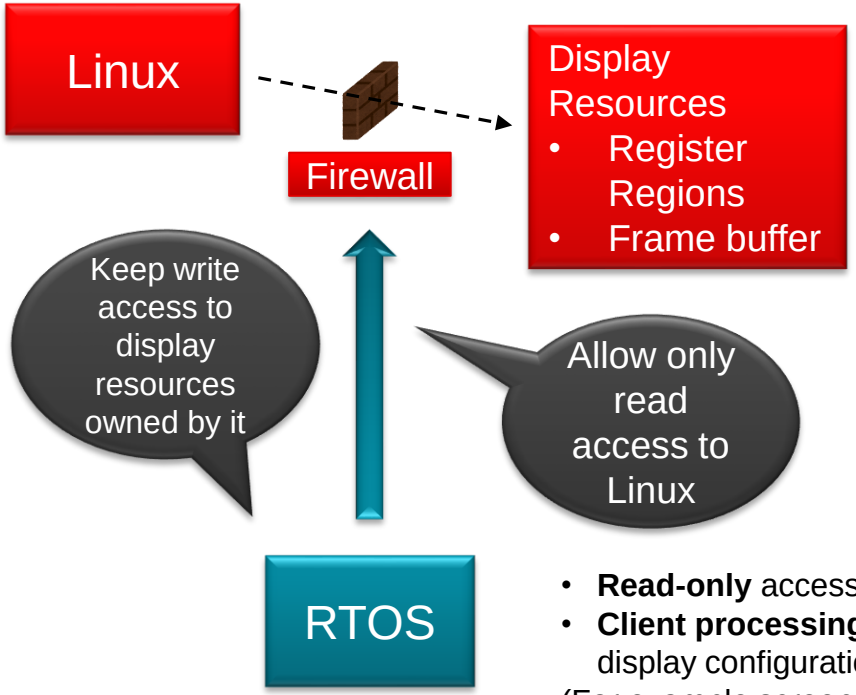
# Example 2: Linux owning one full display pipeline involving video port, overlay manager and video pipeline.




This can be supported with below configuration :

```
&dss0 {
        ti,dss-shared-mode;
        ti,dss-shared-mode-vp = "vp1";
        ti,dss-shared-mode-vp-owned = <1>;
        ti,dss-shared-mode-common = "common1";
        ti,dss-shared-mode-planes = "vid";
        ti,dss-shared-mode-plane-zorder = <0>;
        interrupts = <GIC_SPI 85 IRQ_TYPE_LEVEL_HIGH>;
};
```

# Firewalling : Expose RTOS configured display mode




```
/*
* For shared mode, with remote core driving the video port, make sure that Linux
* controlled primary plane only enumerates video port screen size resolution set by
* remote core
*/
if (dispc->tidss->shared_mode && !dispc->tidss->shared_mode_owned_vps[vp_idx]) {
        struct drm_display_mode *mode = NULL;

  int vp_hdisplay = VP_REG_GET(dispc, vp_idx, DISPC_VP_SIZE_SCREEN, 11, 0) + 1;
  int vp_vdisplay = VP_REG_GET(dispc, vp_idx, DISPC_VP_SIZE_SCREEN, 27, 16) + 1;



  if (mode->hdisplay != vp_hdisplay ||
      mode->vdisplay != vp_vdisplay) {
      dev_err(dispc->dev, "%dx%d doesn't match with VP screen size %dx%d in shared
                  mode\n",
      mode->hdisplay, mode->vdisplay, vp_hdisplay, vp_vdisplay);
          return MODE_BAD;
      }
}
```

- The processing core using the video port **in shared mode** can configure its **owned video pipeline** by reading the video port registers.

- **Read-only** access to **shared** resources
- **Client processing cores** to get a picture of current display configuration.

(For example screen width, screen height)

# Why we took static partitioning approach?

- Leveraging hardware features :
  - Supports robust partitioning of display resources.
  - Provides interference free environment.
  - Allows parallel and independent control of display resources.
  - Duplicated interrupts to each core
- No extra processing overhead or time delays due to IPC or processor context switch.
- Helps achieve shorter SW development cycle compared to IPC.

# Demo : Crashing HLOS ( TI AM62P SoC)

# Demo

- Booth : E-D /P3

# References

- RPI-FKMS : drivers/gpu/drm/vc4/vc4_firmware_kms.c
  - https://github.com/raspberrypi/linux.git
- https://events19.linuxfoundation.org/wp-content/uploads/2017/11/Display-Virtualization-with-KVM-for-Automotive-Systems-ALS-Laurent-Pinchart.pdf
- AM62p technical reference manual
  - https://www.ti.com/lit/ug/spruj83/spruj83.pdf?ts=1712549605347&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FAM62P-Q1%253FkeyMatch%253D%2526tisearch%253Dsearch-everything%2526usecase%253Dpartmatches
- RPMSG KDRV : https://git.ti.com/cgit/ti-linux-kernel/ti-linux-kernel/tree/drivers/rpmsg-kdrv?h=ti-linux-4.19.y
- Automotive display cluster on TI platform: https://www.youtube.com/watch?v=I1E7yzAadD4
- Crashing HLOS: https://www.youtube.com/watch?v=WIcds6HGeMI&t=884s
- Static Partitioning : https://lore.kernel.org/all/20240116134142.2092483-1-devarsht@ti.com/

# Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.

# Q&A

- Contact Information:
  - Devarsh Thakkar devarsht@ti.com
    - https://www.linkedin.com/in/devarsh-thakkar-541a6a49/
  - Soumya Tripathy s-tripathy@ti.com
    - https://www.linkedin.com/in/soumya4779/

## Learn more about TI products

- https://www.ti.com/linux
- https://www.ti.com/processors
- https://www.ti.com/edgeai

Why choose TI MCUs and processors?

**Scalability**

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

**Efficiency**

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

**Affordability**

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

**Availability**

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.

# Demo : Display sharing ( TI AM62P SoC)